J2EE-Praktikum

JNDI

Peter Thiemann

Universität Freiburg

J2EE-Praktikum, WS2005/2006

Inhalt

Java Naming and Directory Interface (JNDI)

Zweck

- Auffinden von Services
- Zugriff auf Services
- Directory Struktur

JNDI ist virtuell

Directory Services können ortsunabhängig verlinkt werden

JNDI ist dynamisch

Treiber für Services werden bei Bedarf geladen

JNDI und EJB

EJB Server müssen JNDI unterstützen

Java Naming and Directory Interface (JNDI)
Verwendungen

JDBC

- `DataSource`-Objekte enthalten Information über eine Datenquelle (z.B. eine Datenbank)
- diese Informationen sind: Namen, Treiber, Ort, etc
- JDBC enpfiehlt `DataSourcen` mit JNDI abzulegen

JMS (Java Messaging Service)

Verwendet JNDI für Zugriff auf

- Konfigurationsinformation
- Message Queues
- Topics

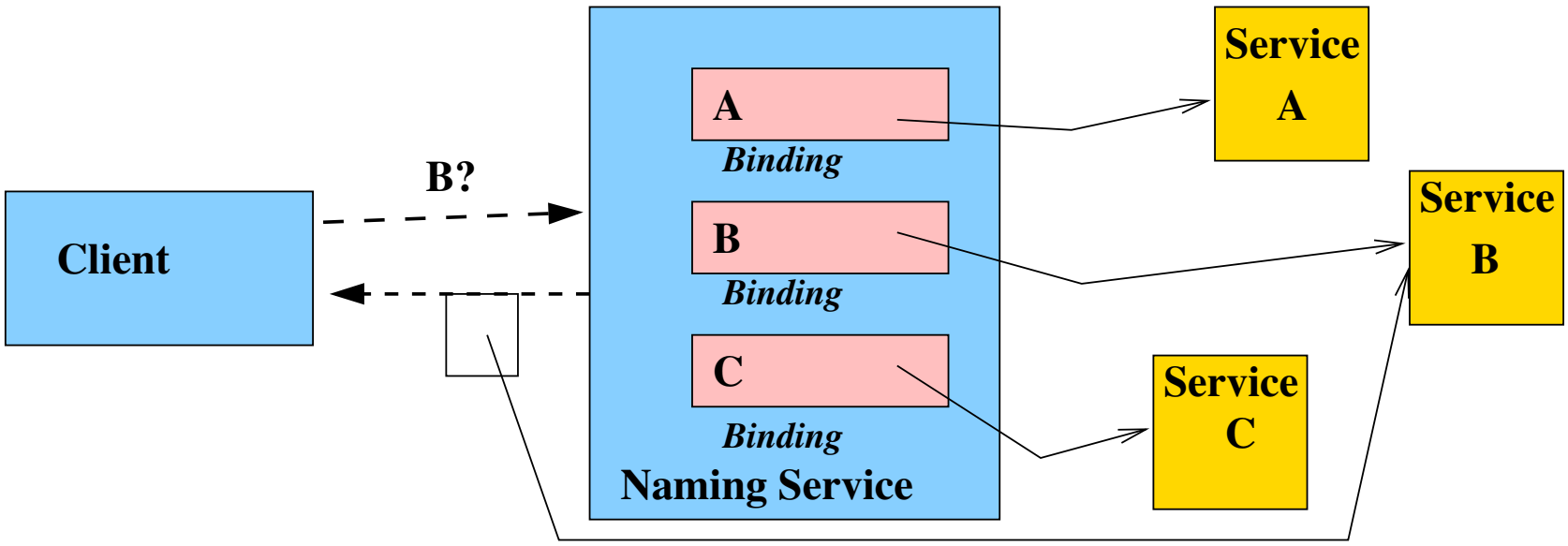
EJB

Zugriff auf Home-Interface mit JNDI

Java Naming and Directory Interface (JNDI)
Vorteile von JNDI

- Eine zentral verwaltete Informationsquelle einfach zu verwalten und zu durchsuchen
- JNDI kann Java-Objekte direkt ablegen

Naming Services



- Naming Service verwaltet Menge von *Bindungen*
- Jede Bindung ordnet einen Namen einem Objekt zu

Naming Services
Beispiele

Common Object Services (COS) Naming

Naming Service für CORBA

Domain Name System (DNS)

- Verteilter Naming Service für Internet Domain-Namen
- Bildet Namen auf Ressourcen (z.B. IP-Adressen) ab

Lightweight Directory Access Protocol (LDAP)

Hierarchische Struktur von Knoten mit je einer Menge von Name-Wert Bindungen

Network Information System (NIS)

Naming Servives für Netzwerkadministration

Naming Services
Anforderungen/Gemeinsamkeiten

- Mindestfunktionalität
  - Definition einer Bindung
  - Zugriff auf Objekt über Name
- Meist nur Referenzen auf Objekte

JNDI

- JNDI ist ein **Interface** für die Mindestfunktionalität
- Keine Implementierung eines Naming Service
- Erlaubt den Zugriff auf eine Implementierung durch einen unterliegenden Naming Service
- Abstrahiert von Unterschieden

Naming Services
Unterschiede

Namenskonventionen

- DNS: `mopo.informatik.uni-freiburg.de`
- LDAP: `cn=Peter Thiemann, o=Uni Freiburg, c=DE`

JNDI

- Repräsentiert Namen durch Klassenhierarchie `Name`
- `Name` besteht aus Folge von Namensteilen
- Methoden darauf sind unabhängig von Namenskonvention

JNDI-Typen
Das `Context`-Interface

- Kontext: Menge von Bindungen mit gleicher Namenskonvention
- `Context`-Objekt erlaubt
  - Erstellen und Löschen von Bindungen
  - Umbenennen
  - Auflisten von Bindungen
- Für strukturierte Namensräume
  - Erzeugen und Löschen von Subkontexten
  - Navigation
- Einstiegspunkt: Klasse `InitialContext`
  - Art des verwendeten Naming Service
  - ID und Passwort (falls erforderlich)

JNDI-Typen

`Context` — Binden und Anfragen

`void bind(String stringName, Object object)`

Binds a name to an object. The name must not be bound to another object. Intermediate contexts must exist.

`void rebind(String stringName, Object object)`

Binds a name to an object. Intermediate contexts must exist.

`Object lookup(String stringName)`

Returns the specified object.

`void unbind(String stringName)`

Unbinds the specified object.

`void rename(String stringOldName, String stringNewName)`

Changes the name to which an object is bound.

JNDI-Typen

`Context` — Auflisten

`NamingEnumeration listBindings(String stringName)`

Returns an enumeration containing the names bound to the specified context, along with the objects bound to them.

`NamingEnumeration list(String stringName)`

Returns an enumeration containing the names bound to the specified context, along with the class names of the objects bound to them.

- Zu jeder Methode gibt es eine Variante mit `Name` anstelle von `String`, damit Programme unabhängig von Namenskonventionen gehalten werden können.

JNDI-Verwendung
Beispiel

- Funktionen im Beispiel
  - Verbinden mit Naming Service
  - Auflisten aller Bindungen
  - Abfragen einer Bindung
- Verwendet *file system service provider* von Sun
  - Sicht auf Filesystem als Naming Service
  - Bindet Namen an vollen Pfad (gewöhnliche Datei) oder an Subkontext (Directory)
- Download der service provider von `http://java.sun.com/products/jndi/serviceproviders.html`
- J2EE Container haben einen “eingebauten” JNDI-Provider, der eine interne Datenbank verwendet

JNDI-Verwendung
Code, Erzeugen des Kontexts

```
Hashtable env = new Hashtable();
env.put(
  Context.INITIAL_CONTEXT_FACTORY,
  "com.sun.jndi.fscontext.RefFSContextFactory"
);
env.put(Context.PROVIDER_URL, args[0]);
Context ctx = new InitialContext(env);
```

Initialer Kontext wird aus einer Umgebung `env` erzeugt, die

- den JNDI Provider und
- die Start-URL (hier `file:///...`)

spezifiziert

JNDI-Verwendung
Code, Auflisten

```
        if (args.length == 1) {
          NamingEnumeration bnds = ctx.listBindings("");
          while (bnds.hasMore()) {
            Binding bnd = (Binding)bnds.next();
            System.out.println(
              bnd.getName() + "␣" + bnd.getObject()
            );
          }
        }
```

Auflisten aller Bindungen, falls keine weiteren Kommandozeilenargumente

JNDI-Verwendung
Code, Abfragen einer Bindung

```
        else {
          for (int i = 1; i < args.length; i++) {
            Object obj = ctx.lookup(args[i]);
            System.out.println(args[i] + "␣" + obj);
          }
        }
        ctx.close();
```

Auflisten der angefragten Bindungen

Directory-Kontexte

Klasse `DirContext`

- Subklasse von `Context`
- Erweiterungen
  - Attribute
  - Subkontexte
  - Suche
- Nur `String` Variante, `Name` Variante existiert auch

Directory-Kontexte

`DirContext` — Attribute

```
void bind(
    String stringName,
    Object object,
    Attributes attributes
   )
void rebind(
    String stringName,
    Object object,
    Attributes attributes
   )
```

Bei `rebind` werden die Attribute der vorherigen Bindung beibehalten, falls keine neuen Attribute spezifiziert sind.

Directory-Kontexte

`DirContext` — Attributabfrage und Erzeugung

```
Attributes
getAttributes(
    String objName
    // String [] whichAttributes
)

DirContext
createSubcontext(
    String stringName,
    Attributes attributes
)
```

Directory-Kontexte

`DirContext` — Suche

```
NamingEnumeration search(
  String name,
  Attributes toMatch // String [] attrsToReturn
)
NamingEnumeration search(
   Name name,
   String rfc2254Filter,
   SearchControls searchcontrols
)
```

- RFC2254: String-Repräsentation für LDAP Suchfilter
- Beispiele dafür

  ```
  (cn=Babs Jensen)
  (!(cn=Tim Howes))
  (&(objectClass=Person)(|(sn=Jensen)(cn=Babs J*))
  (o=univ*of*mich*)
  ```

Directory-Kontexte
SearchControls

```
SearchControls(
  int      nSearchScope,
  // OBJECT_SCOPE, ONELEVEL_SCOPE, SUBTREE_SCOPE
  long     maxEntriesToReturn,
  int      maxTime,
  String[] attributesToReturn,
  boolean  returnObjects,
  boolean  dereferenceLinks
)
```

Namen

- `Name` (Interface) definiert Operationen auf Folge von Namensteilen
- Kann gemischt (*composite*) oder zusammengesetzt sein
  - composite: Teile aus verschiedenen Namensräumen
  - compound: hierarchische Benennung
- Komponenten sind nummeriert $[0, N)$
- Komponente 0 ist an der Wurzel der Hierarchie

Zusammengesetzte Namen

- `CompositeName`
  - Jede Namenskomponente ist `String` aus einem Namensraum
  - Für hierarchische Namensräume kann eine Komponente weiter aufgeteilt werden, durch Parsen des Strings. (`CompoundName`)
  - Standard-Stringrepräsentation: `/`-separierte Namenskomponenten
  - Namenskomponenten können beliebige Strings sein (auch leer)
- `CompoundName`
  - Definiert Parser für einen Namensraum
  - Konfiguriert durch verschiedene Properties

Zusammenfassung

- JNDI ist eine generische Schnittstelle für Namens- und Directory-Services
- Service-Provider für LDAP, Filesystem, DNS, . . .
- Integriert mit Java
- Abstraktion von konkreten Namensräumen